IT 09 062
Examensarbete 30 hp
December 2009

Automatic error diagnostic for network connection problems

Christer Folkesson

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Automatic error diagnostic for network connection problems

Christer Folkesson

Customer support inquires about software and network settings takes time and costs money, both for the customer and the vendor giving the support. Doing troubleshooting and problem solving automatically can lower the need for inquires, and when needed, raise the quality of the contact for the customer.

Can a nearly automatic program do sufficient network diagnostic and let an end user alleviating the problem found all on their own?

I surveyed the offerings of such functionality by the current desktop operating system in use. I also developed a prototype that could do automatic error diagnostic of network software settings and the connection to important network services.

The newest operating systems (Windows 7 and Mac OS X Snow Leopard) have good automatic error diagnostic facilities for generic network problems, but older versions like Windows XP has only limited capability. The prototype look at the same problem, but also take it a step further and test the availability and connection quality to a vendor's specific network service.

For solving generic network problems and when having the newest version of the operating system the benefit of a separate tool is not great. But when a company requires more specific network testing related to their product, and possible retrieve other valuable information for the customer support about the computer setup, a custom developed tool has its benefits.

Handledare: Jan Berg
Ämnesgranskare: Arnold Pears
Examinator: Anders Jansson
IT 09 062
Tryckt av: Reprocentralen ITC

Preface

I have done this master thesis project for the computer science degree at the Department of Information Technology at Uppsala University. The thesis work has been done at bwin games AB in Stockholm, Sweden, where Jan Berg has been my supervisor. I have not co-operated with any other student on this project. The work started in Mars 2009 and was finished in late September 2009.

There are some people I like to thank in no special order.
Jan Berg for being a good supervisor, not looking over my shoulder all the time but often available when I needed some help.
Dr. Arnold Pears, my reviewer. Without his help I would not have found the BART algorithm and his very well appreciated help with finalizing this report.
Marie-Sofie Karlsson for her introduction to the software development system environment and side coach at bwin.
Benjamin Lloyd for his input on the customer service department's behalf.
Christoffer Lernö for giving feedback on parts of my code with great enthusiasm and teaching me about software testing frameworks.
Dr. Niklas Pettersson for his very valuable feedback on my project presentation (the first one, so I could improve the later presentations).
Sohail Sahab for using his skills in human computer interaction to evaluate the graphical design of the prototype.

And to the rest of the people at bwin for making the work very fun!

Contents

List of Tables

List of Figures

1. Introduction

Although computers have become easier to use and handle, the average user's technical knowledge of the computer has gone down as now also 'normal' people and not only technology 'geeks' use them. Thus many people need help when there is some malfunction as now everyone is not interested in knowing everything there is to know about a computer. The 'normal' people just want it to work. So this leads to need of customer support. Such support over telephone or via e-mail is common but it is tedious for the customer service agent to understand the problem and correct it while only communicating by voice or text and not see the content of the computer screen. It would be of great value to have a tool that can do routine troubleshooting and tell the user and the support personnel what the problem actually is without having a support agent guide the user through the entire troubleshooting process.

Apart from being a good service to the customer to avoid some support call and raise the quality of others it is also important to companies like bwin that has a product which generates continues income. A software company producing off-the-shelf applications with a one-time payment at time of purchase is not so dependent on that the customer can use their software – they got their money already. For companies in online gambling business it is vital that the customers can continue to play as they generate the income *when* they are playing.

2. Glossary

ARP (Address Resolution Protocol) – is a method for retrieving a host's link layer (hardware) address by providing the network layer (often IP) address.

Bwin – The online gaming and online sports betting company sponsoring this master thesis.

DHCP (Dynamic Host Configuration Protocol) – Standard for automatically configure network settings on hosts on a network instead of doing it manually. If the DHCP server is unavailable when a host requests settings it will become somewhat like an orphaned child, one possible cause of loss of the internet connection.

DNS (Domain Name System) – Is a distributed database mapping names to IP addresses. If all of the configured DNS servers of a host fail it can not contact other hosts (e.g. web servers) via domain names, only by directly providing the IP address.

ICMP (Internet Control Message Protocol) – Is a core protocol in the IP suite. It is not used for user/application data transfer, instead for error messages to help the network function. One notable function is to send a "ping", meaning to transmit an "echo request" (type 8 message) and

expect a "echo reply" (type 0 message) in return. The sender measures the round trip time of this.

JNI (Java Native Interface) – Allows java code running in the java virtual machine to communicate with operating system specific native code, both to call and be called by.

JNLP (Java Network Launching Protocol) – specifies how a java web start application should launch. It defines a XML schema to this end.

JRE (Java Runtime Environment) – is composed of the java virtual machine (JVM) and class libraries. JVM executes java bytecode and the class libraries implement the java application programming interface (API).

Native code – Opposed to Java's platform independent source code and byte code (the code that the JRE understands) native code mean source code and compiled code aiming at a specific platform (operating system and instruction set architecture).

NetBIOS (Network Basic Input/Output System) – is an API aimed at providing services on a network. There are several adaptions using different network protocols as carrier.

Ongame network – The poker network platform that is owned and maintained by bwin.

Operator – A company that has a branded poker client developed by bwin and have their own 1st line customer service but which infrastructure in the poker network is hosted on the Ongame network. There are around 25 operators not counting bwin's own brand on the network.

Test – When referring to the prototype it means a plug-in that does some investigation of the computer or related things, e.g. the network connection test.

Tool – When referring to the prototype it means a plug-in that does some "work" without the intent to investigate anything.

WINS (Windows Internet Name Service) – is a Microsoft implementation of the name service for computer names offered by NetBIOS.

3. Problem Description and Rationale

Internet is a very complex "machine" and it is bound to have operation disturbances. Internet service provider companies can monitor disturbances in their own network but for laymen this kind of information is not available.

To correct network problems it takes often quite bit of experience and most users never get the time to learn about this topic. But unlike many other fields the computers has a very good for doing automatic repair of its own – the computer can be controlled by programs and not only humans.

Network diagnostic is one field where many repairs can be performed by the computer but some kind of repairs need the ability to affect objects in the physical world. Here an interaction is required to guide the user.

Functionality for more-or-less automatic network diagnostic, to find and correct network problems, is already implemented in modern versions of

desktop and laptop operating systems like Microsoft Windows and Apple Mac OS X. But it is not always the case that they provide enough information about the problem.

The master thesis specification is included in Appendix A, where the goals and delimitations are explained in greater detail than here in 3.1 and 3.2. The specification was created by me to define more precisely a very short master's thesis proposal and with approval and suggestions on every part by Jan Berg at bwin Games.

3.1 Goals

The goals of this master's thesis are the following.

- To investigate how network connection problems can be diagnosed and what facilities are provided by today's operating systems.
- Find out what problems that are common among the customers, both network-wise and others. Are there some general issues or are the issues specific to bwin's software?
- Identify the most important areas of functionality for a diagnostic tool running on the customer's computer.
- To develop a prototype of diagnostic program that is aimed to the average customer for diagnosing network problems. A second aim is to incorporate more functionality from suggestions in previous surveys. The application should be implemented with Java as an application that can be retrieved via Java web start. The focus is on network connection problems, other possible suggested needs are secondary.

3.2 Delimitations

The prototype should be made to help inexperienced users with network related problems that they may encounter. The aim is not at network professionals and their methods and tools for solving complex network problems. Only automatic ("press a button") diagnostic is required of the prototype, no need to attempt to "repair" a computer's perhaps faulty network configuration settings.

4. Work process of this master thesis

The preparation of the this project began in late February with myself and Jan Berg sharing ideas over what goals the project would have and in the end I had written a proposal for a master thesis project specification (Appendix A).
After that I started my work in mid-March with preliminary reading on the subject, went on with first contact with the computer network people and customer service at bwin. Meanwhile I was experimenting with the different technologies (e.g. Java web start) that I would use. In April I started the first step of the prototype while continuing reading about the subject. Work

continued, with milestones in mid-May and mid-June. There were some presentations at bwin of these milestones to show what was done and to get feedback. It was at this time in the middle of the summer I found the BART algorithm and shifted focus to understanding it and implementing it which continued after the summer vacation. Ca. 1/3 of this report was written while working on the prototype, the rest in the weeks I had put aside in the end of the project when only minor changes was made to the prototype. In late September I held the final presentation of the project at bwin and at Uppsala University.

5. Methods for Diagnosing Network Problems

5.1 Tools in Today's Operating Systems

To develop a prototype was not the only goal with this project, also surveying the current offering of such functionality from the operating systems' vendors was important – perhaps it would be superfluous to develop this kind of program for real world usage.
Here I present the current status of network diagnostics for the major desktop/laptop operating systems on the market.

5.1.1 Windows XP

Windows XP can only inform the user that the cable is not connected (or that the interface is not associated with a wireless network) or that it has "limited or no connectivity", often reported when it has not gotten an IP address from a DHCP server.
Microsoft introduced the "repair" function with Windows XP. When a network connection is not working it does a number of things to try to fix the problem. But it does not present possible causes if it fails, it only tells the user at which point it failed. These are the steps that are performed while repairing [1]:

- Transmitting a DHCP broadcast message asking for address *renewal*. Note that it does not *release* the current address first as that could worsen the situation if the computer does not receive a DHCP lease response from any server.
- Flushes the ARP cache, which is the cache of previous learned mappings between IP addresses and network interface addresses (MAC addresses).
- Flushes the NetBIOS name cache and reloads static entries.
- Re-register the computer with a WINS server (if such exists).
- Flushes the DNS cache, the mappings between IP addresses and fully qualified domain names.
- Re-register the computer with a (dynamic) DNS server.

There is also another tool from Microsoft called "Network Diagnostics for Windows XP" [2] released for free in 2006. This tool purports to do the following tests:

- Test the IP address configuration.
- Test the default gateway.
- Test Winsock.
- Test DNS.
- Test software firewall.
- Validate the internet connection.

After testing it can do some repairing actions or suggest to the user what to do. It provides a log file with much information about what has been tested and the results.
However it has at least one drawback. When using two network adapters it stopped testing and just directed the user to customer support with the warning message "This machine has more than one Ethernet or more than one Wireless adapter". I think that is an unnecessary limitation on the software.

The introduction of the repair function was a significant improvement compared to not having any. But it lacks functionality with which to inform the user about possible steps to take if it is unsuccessful in the repair process. The separate tool had some of this functionality. Also it is not system-wide; the user has to select a specific network connection (which needs to be enabled first). As there are some strange "network connections" that sometimes are shown in the network connections list (i.e. IEEE 1394 and "Microsoft TV/Video connection") the user might select the wrong one when trying to repair it. This level of technical competence should not be expected of the average user.

5.1.2 Windows Vista

Vista has a slightly more refined network diagnostic suite. The user does not need to begin by selecting a specific network connection; instead there is a "diagnose and repair" link in the "network and sharing center". It tries to find the problem and repair it. If correcting the connection state requires a physical presence (e.g. "connect the network cable" or "restart the home router") the software gives appropriate instructions to the user.

5.1.3 Windows 7

In Windows 7 the diagnosis of network problems has been reworked. A "Windows troubleshooting platform" has been developed to support many different tests (not only network related) including option to extend it with self-made tests.
In the troubleshooting guide the user has to select which test to run, but the choices are easy to understand. Namely, the user can choose from the

following tests to run from the "network and sharing center" (where only the network related tests are displayed):

- **Internet Connection** – tries to reach Microsoft's web page or a page provided by the user.
- **Shared Folders** – the user provides some network file share where a problem exists that requires software guided diagnosis.
- **Home Group** – troubleshoot Microsoft's improved ad hoc network service in windows 7 that gives streamlined access to resources on other windows 7 computers at home.
- **Network Adapter** – if nothing is working this is the guide to start with, it diagnoses the network adapter's settings etc.
- **Incoming Connections** – checks things regarding the software firewall so programs or services can work.
- **Connection to a Workspace Using DirectAccess** – DirectAccess (written without a space between the words) is a VPN technology based on IPv6.

5.1.4 Mac OS X – Leopard & Snow Leopard

OS X has a similar network diagnostic as Windows 7. The user has to select which network adapter to test and then it reports status for a few different items that are labeled Ethernet, Network settings, ISP (internet service provider), Internet and lastly, Server. When testing wireless connection the Ethernet label is replaced with Airport and Airport settings.

5.1.5 Ubuntu Linux 8.10 – Intrepid Ibex

There does not seem to be any *automatic* network diagnostic utility shipped with Ubuntu. There is a network tool that is just a GUI frontend to traditional programs like ping, traceroute, nslookup etc. The operating system's help files contain some troubleshooting guides for networking.

5.2 Possibilities to Diagnose From a Single Network Host

First step in diagnosing a connection is to see if there actually is any connection at all, thus testing *reachability*. But this is not sufficient in itself, the user most probably already knows that there is a problem. To help the user alleviate the problem the program, needs to provide information about the (probable) location or cause. The key points are:

- **Physical connection** – is the network cable plugged in or is the wireless network card associated with a wireless network?
- **Presence on the network** – does the machine have an IP address at all?
- **Inter network capability** – can the default gateway be reached?

- **Vicinity** – try different sites on internet, if none of them works the problem is probably close to the default gateway, like in the internet service provider's network.
- **Destination** – can the ongame network's servers be reached?
- **Name resolution** – perhaps the DNS server (-s) used by the client has a problem with resolving the name of the game servers. This problem differs from the others, since in this case there is no problem with the direct path between the client's computer and operator.

After the above established criteria are verified to work, the *quality* of the connection should be determined – reaching the game server a single time is not sufficient. Factors that play a role in the quality of a connection are:

- **Message loss** – if a data packet gets lost en route it can sometimes be detected with an ICMP message (type 3 destination unreachable and type 12 time exceeded), but still it has to be retransmitted and then the measure of quality goes down and the total time for the message to reach the server increases. If a program is not written robustly message losses can make it work very poorly.
- **Round trip time** – how long time does it take for a message to reach the server and a response to travel back to the sender under normal working conditions? If it takes too long time the program will feel unresponsive. Luckily online poker gaming is not as sensitive to this as an action multiplayer video game.
- **Bandwidth** – how many bytes of useful data can be sent and received within a given timeframe?
- **Message integrity** – the message itself should not be altered en route. This problem is not so common thanks to automatic error detection, but when a message gets corrupted the error can-not be corrected but instead will be retransmitted. In some situations this situation can be misinterpreted as a message loss problem.

5.3 Possibilities to Diagnose Between Two Network Hosts

Internet service providers can passively monitor the amount of traffic that passes through their routers and thus find a bottleneck in their network, but such information is not available to the general public. So to measure the available bandwidth across the internet two hosts need to cooperate in some way. It can be as simple as downloading a large file from a server to see the speed of the path between them. But this is generally a bad idea for at least three reasons:

- If either the client or the server pays per transferred data it can cost a lot in the end, especially for the server owner if a lot of clients test their bandwidth in this way.

- The server can become congested by all the bandwidth testers, so the bottleneck is always at the server. The goal is to measure the speed on the path between the hosts, not to create a bottleneck at one of them.
- A client does not want to fill (possible all) their available bandwidth for a test, it creates disturbances, e.g. using streamed real-time media at the same time would probably not work (unless some quality of service is employed).

It seems common internet sites for speed testing employ the method of filling the available bandwidth and measuring per second throughput. In *Figure 5-1* the result is shown from running the bandwidth test on the internet site www.speedtest.net and on www.bredbandskollen.se. Notice that the test is conducted on a gigabit Ethernet connection, so even though the highest usage was only 25% that is 250 Mbps.

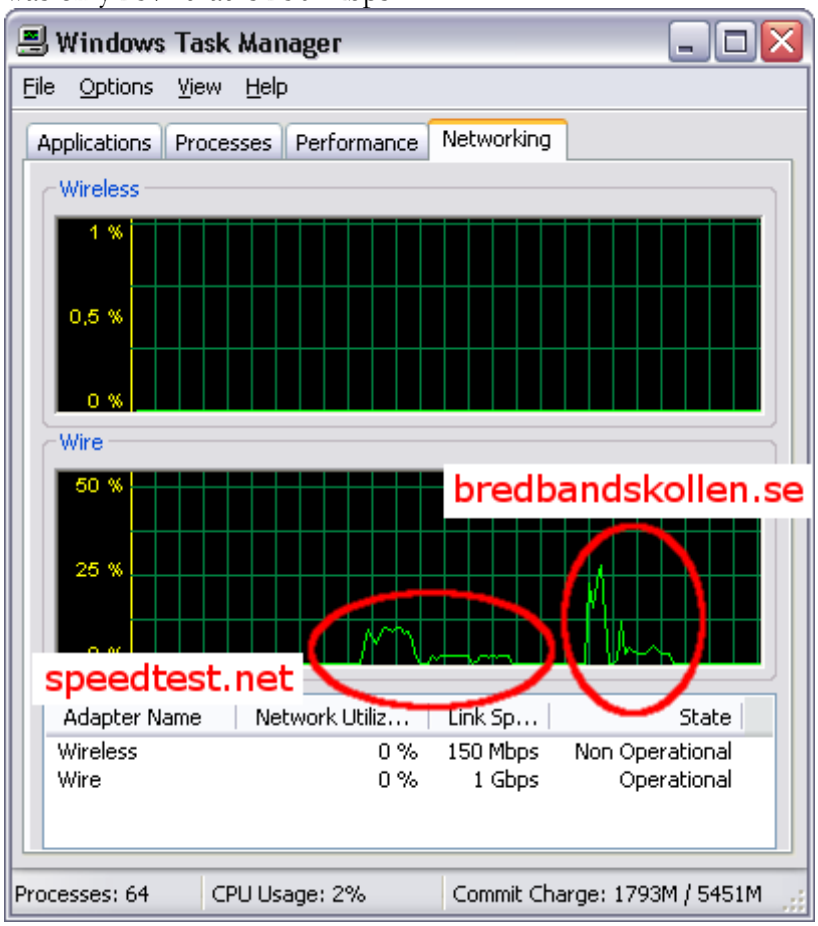

Figure 5-1. Bandwidth usages during test of speedtest.net and bredbandskollen.se.

Thus we want a better way to measure the available bandwidth. Eklin et al. (2006) [3] proposed an algorithm they call "BART". That algorithm is the only one I have studied, but there are others of this kind, e.g. pathChirp [4].

5.4 BART (Bandwidth Available in Real-Time)

BART is a method that analyzes the currently available (unused) bandwidth in a packet-switched network. The algorithm works by sending a series of network packets with a known time delay between each packet (the rate). It tries to find the minimum amount of time separation where packets arrive at the same time separation as they were sent. If the packets arrive at a lower rate it means that they have experienced traffic congestion on some hop en route. The hop that had the biggest congestion is the overall bottleneck of the whole route. There amount of packets that needs to be sent to get a good estimate are few hence the BART algorithm solves both previously stated problems about bandwidth testing.

The algorithm analyzes the collected data after each "packet train" has been completely received and updates its bandwidth estimate on the go. There are other methods that works by the packet train principle but they do offline estimates, meaning that the results are only presented when all the sampling is complete, by which time the estimate perhaps can already by outdated.
The Kalman filter algorithm is used in BART to get a nice average of all the measurements (even though it can be of a very short time scale).

5.5 Kalman Filter

The Kalman filter is a recursive filter-algorithm for estimating the state of a linear dynamic system based on a series of noisy measurements. It is used in a wide range of applications, usually very some sensor reading of the real world is not totally accurate and the process that is measured also has a degree of uncertainty. A good first guide to the Kalman filter is provided by Welch and Bishop (2001) [5].

6. Common problems for customers

Here are the most common difficulties for customer service department when customers get in contact with them when they have experienced a problem.

6.1 Disconnected while playing

The single biggest problem according to the customer service division occurs when a player gets disconnected for an unknown reason while playing and loses his (good) poker hand and possibly a chance to win in a tournament. The difficulty is to know where the error occurred, if it was the player's internet connection causing the problem or at the game servers, in which case bwin would take responsibility for any losses and issue refunds.

6.2 Location of log files in the file system

The poker client stores its log files in a location that is the correct path for this kind of information according to the design guidelines [6] on a Windows

operating system, but this directory is hard to find for a novice customer. The folder location in question is the local application data folder of the windows user's profile. On a normal Windows XP installation (Vista and 7 stores this information somewhat differently) this folder is located in
"C:\Documents and Settings\<*username*>\Local Settings\Application Data" but it may differ as an administrator can change this path. On top of this the last two folders, "Local Settings\Application", have the hidden folder file system attribute. Many users do not show hidden files and folders in Windows Explorer.

6.3 Log file content and size

Currently it seems that the content of the log files is not so helpful when trying to find a solution to a disconnection problem. Partly because of lack of information to determine the error, and partly that parts of it can be hard to understand for people that have not developed the system (such as the customer service agents).
E.g. on Windows the error codes given by the windows sockets (winsock) object when a network socket has been abruptly terminated give quite good information about the loss of connection, but this is not recorded.
The logger produces quite large log files in terms of bytes, this makes the log files hard to handle as it is normal procedure for the customer service agent to request the user to send the log file, but some e-mail systems can have a limit of a few megabytes for attachments.

7. System Design of the Prototype

In the planning stage for the prototype, after realizing certain limitations, I came up with a simple division between different components of the program. It stayed the same during development and is illustrated in *Figure 7-1*. The Core works as a provider for different testing-extensions. To assist it, it has the JNI subsystem where one can have special extensions that also provide functionality to testing-extensions. The "Test & Tools facility" organizes the extensions. The testing-extensions provide their own GUI.

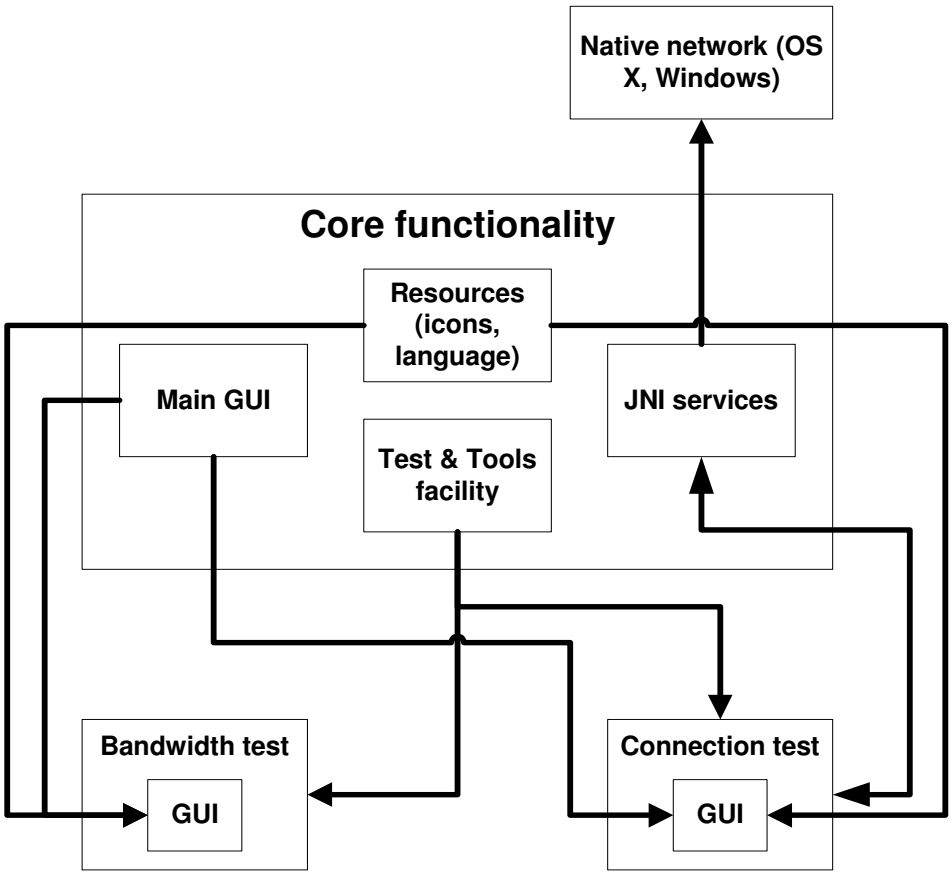

Figure 7-1. Basic component diagram for the prototype

7.1 System Architecture

The requirement of the prototype from the thesis specification was that it should be extensible, so that new diagnostic tests can be added in future versions without extensive refactoring of the code. The plug-ins can be programmed to do a variety of things, not only diagnostic tests, but also tools where the task is not to find something (e.g. find a network error) but to do something (e.g. retrieve the log files). This "plug-in" architecture is built to be extensible at compile time, not at run time. There is no sandboxing of the plug-ins as they are implemented by the same party that develops and distributes the software, thus there are no security considerations that would require a sandbox environment. In contrast to the situation for a web browser with its plug-ins.

But to not re-invent the wheel every time the core of the program also contains general services offered to the plug-ins. These services can be upgraded in future versions, and thus the plug-ins are not totally independent of the version of the core program, but this is not a problem, since the user never have the opportunity to install the plug-ins themselves. This includes the network access facility customized for each operating system but exported through common interfaces.

7.2 Graphical User Interface

As the target audience for the prototype is a novice to average skilled computer user it has to be simple and obvious what can be done with it. It should not be required of the user to study a user manual; instead everything needs to be intuitive.
As it is built as a plug-in architecture with distinct tests and tools it is natural to create a list of them in the main window, from where the user can start a test or a tool.
Plug-ins provides their graphical user interface; it is not something made available through the core program.
Another aspect of user friendliness is that not everyone understands English. Thus the prototype has support for different languages that automatically is detected and used. The only implemented languages are Swedish and English. The latter is default for everyone that does not have Swedish as regional setting in their operating system.

7.3 Distribution model

The master thesis calls for a Java application. It should be in a single JAR-file and when signed (applied a digital signature) it can be distributed via Java web start and do privileged operations, not being sandboxed like ordinary applets or web start applications.
Java web start is a distribution method where a HTTP server hosts a `.jnlp` file with a custom internet media type (MIME type) called `application/x-java-jnlp-file`. When requested by a web browser (typed into the address bar or referred via a link) it will automatically check if the right version of the Java runtime environment is installed. If not, it offers the user to install it. After this it downloads the application and installs it. The advantage is that the Java runtime environment can be "bundled" with the application; one does not have to request the user to visit some other web site to download a Java runtime environment.

7.4 Input on the System Design from other parties

As the primary interest of a tool such as this within the company is at the customer service group, they were asked about what functionality that they would like to see in a tool for doing diagnostics on customer's computers run by the customers. The inquired people were 2nd line agents but several of them had worked as 1st line agents and back then had contact directly with customers. As the prototype of the master thesis was not meant to be developed only by their wishes, it would not be mandatory if it didn't fit well with what the thesis specification mandated of the prototype and also not to make only routine programming work. Actually the most sought specific feature was a way to help the customers retrieve the log files from the poker client, as the customers had a hard time finding them; this could possible give

more information when trying to uncover what went wrong. It is extremely easy to program but is of too limited scope to be an appropriate focus for a master thesis project. Input that I did incorporate was to make the graphical user interface very simple and with a bigger font size than normal.

8. Implementation of the Prototype

8.1 Plug-in Architecture

All tests implement a Java interface with two tasks: to identify it and provide an object implementing the *runnable* interface. Platform specific native code is not directly exposed to plug-ins; instead a special package takes care of that with a uniform interface.

8.2 Network Diagnostic Test

Most parts of this test require functionality that only can be provided by the custom made native code for each platform, c.f. with the section 8.5.1 *encountered impediments - Java* below for the specific issues.
Here each stage is explained and a brief mention about with what it was implemented on each platform. Each stage's corresponds with the numbers in *Figure 8-1* in order to make it easier to follow the text.

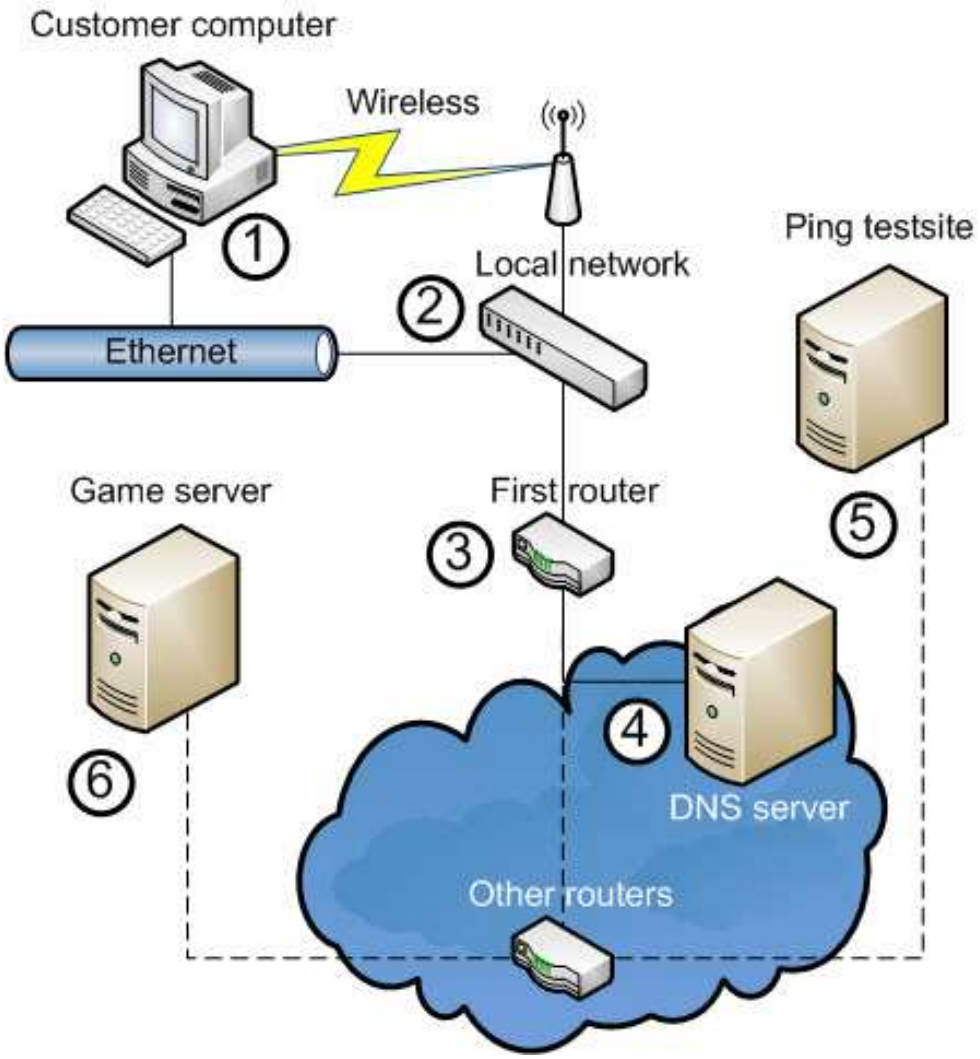

Figure 8-1. Illustrating where the different stages of network connection tests takes place.

8.2.1 Step 1 – Medium Present

Check if the network cable is plugged in (both ends of it) or if it is a wireless network card it checks if it has been associated with any network. Implementation:
On Windows the native code uses *isNetworkAlive* function exported from the System Event Notification Service (*Sensapi.h)*.
Not implemented on OS X due to time constraint and lack of documentation, c.f. with section **Future development.**

8.2.2 Step 2 – Valid IP Address

Check if the computer has any IP address assigned to it, no matter if it is static or dynamic, except the loopback address or a self-assigned dynamic address. The Java function *InetAddress.getLocalHost()* is used although it only can return 1 address, even if the computer has several assigned. This leads some problems.

E.g. the computer has two network interface cards, a wired and a wireless network. It can be that both have their medium present and both use dynamic IP address assignment (DHCP) but only one has been assigned a "real" address and the other has a self-assigned 169.254.x.y address. In this situation it is possible for the Java function to return the self-assigned address. This can give a false negative test result in this step. However, due to time constraints I have chosen to use this method instead of implementing it from native code where all addresses could be retrieved.

8.2.3 Step 3 – Default Gateway

Check if the computer has a default gateway configured, meaning that it knows the address of the first hop to any other host that does not share the same IP network address. The default route, called "0.0.0.0", in a routing table has the default router as the next hop by definition.
For this test some native code was needed.
Windows: *GetBestRoute* function exported in the IP helper API (*iphlpapi.h*) gives the best route to an IP address.
OS X: *Sysctl* had to be used to retrieve the routing table from the system and then manually find the row referring to the default route and extract the next hop column from that table.

8.2.4 Step 4 – Name Server

Check if the computer has any IP address to some DNS server and test if it responds. Native code for this:
Windows: *GetNetworkParams* function exported in the IP helper API (*iphlpapi.h*) return various parameter about the network configuration, among those is a linked list with DNS servers.
The shell command *scutil* is executed which can list DNS servers and the output is grabbed by the application.

The actual test of the server is straight forward. A datagram package is constructed, that conforms to the DNS specification [7], and sent via UDP. It is a simple query and as the content of the response (what the name resolved to) is not interesting, it only checks that the right ID marker exists in the returned package. The rationale for testing reachability with the DNS protocol instead of ICMP echo requests is that ICMP might be blocked, or the actual DNS server software is not running but the server hardware and operating system works at the given time.
As Windows and OS X are very similar in terms of socket programming (heritage from BSD sockets) and since Java doesn't support unsigned data types (which makes it harder to construct the package where one needs precise control over which bits are set and which are not) the implementation of this is done in native **C** code. Both platforms use the same methods and when the code differs slightly preprocessor directives are used to select between Windows and OS X code.

Java has methods for resolving names built into the standard library. Those records can have been cached previously. The only reliable way to really know that the request was sent to the server is to do it "by hand".

8.2.5 Step 5 – Connection to the Internet

This test sends ICMP echo requests ("ping") to a number of hosts on the internet. The addresses' names imply that they are intended for anyone to use as ping sites. Sites used are listed in *Table 8-1*.

Table 8-1. Servers used for ICMP echo requests.

Host address name	Location
ping.sunet.se	Sweden
ping.port80.se	Sweden
ping.bahnhof.se	Sweden
ping.copper.net	USA

8.2.6 Step 6 – Connection to Game Servers

Check if the program can connect to the game servers. As ICMP echo request can be blocked a stream socket (over TCP) is opened to the game server software that the poker client usually connects to (with the same port). As TCP stream sockets do handshaking this tool will know that the game server is reachable. After the handshake is completed the stream socket is closed down, no testing using the poker protocol (e.g. login procedure) is performed.

8.2.7 The Quality of a Network Connection

The result from the above steps in the test mostly makes sense to a professional. To help a novice user to understand all this the combined result is presented with a "score" (not a competition "score" but a judgment) which is presented both in text and symbolically, on a scale from green to red depending on the score.

The score is calculated from all steps in the network connection test. Steps 1 through 4 have results that are Boolean, either it works or not. Thus they can terminate the connection test immediately. Steps 5 and 6 reduce the score, but the testing process continues.

The actual score is maintained internally as a double precision floating point value. It starts out as 1.0 and decreases with falling connection quality. The final quality judgment is based on this number; it is mapped to a textual score. The levels are listed in *Table 8-2*.

Table 8-2. Quality levels and their required minimum quality value.

Label	Minimum
Perfect	0.95
Good	0.85
Acceptable	0.75
Poor	0.55
Unusable	0.35
No connection	N/A

The ICMP echo requests (the ping test) has associated rules that assist in computing the score. I have constructed these formulae myself and they are not based on any underlying theory. They have been tuned by testing so that the result seems quite fair. In particular the latency "punishment" function was tried many times to come up with a good mix of being forgiving if the latency is not to high but severely bring down the quality score if the latency was high.

If there has not been any response from a site the following deduction is made

$$quality := quality - \frac{0.3}{|sites|}$$

, where *sites* is the set of test sites used.

By dividing the "punishment" by the number of sites used in the test it is more fairly balanced, the algorithm will improve if more reliable test sites are added.

If instead the test computer has gotten at least one response but not all it will do the following deduction and continue to the last deduction

$$quality := quality - 0.5 * \frac{\left(\frac{failed}{total}\right)}{|sites|}$$

, where *failed* are the number of requests without any received response and *total* is the total number of requests sent to that site

For each test site with an average response (*latency*) above 50 milliseconds the following deduction is applied

$$quality := quality - \frac{f(latency)}{|sites|}$$

, where *latency* is the average response time for each particular site and the function *f* is the following

$$f(x) = 0.5 * (0.001 * (x - 50))^3 + \frac{x - 50}{5000}$$

A plotted graph for the function is given in *Figure 8-2*.

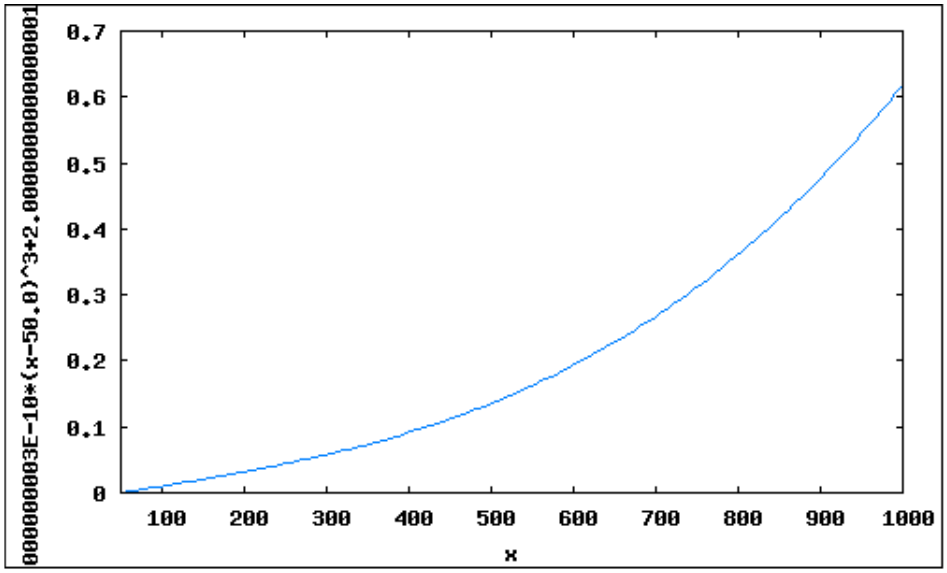

Figure 8-2. Latency "punishment" function's plot. X-axis is measured in milliseconds and Y-axis is the quality penalty value.

8.3 Available Bandwidth Test

The bandwidth test is not integrated with the network connection test due to uncertainty about how well it would work – if it does not measure up correctly there is no reason to have it in the multi-step network connection test. Instead it is listed as a separate test on the main menu of the application.

The goal of this part of the project, implementing an efficient bandwidth test, was only to see if I could do it and test such a method, not to integrate it coherently with the rest of the prototype, which is a much bigger task. It was sufficient as it is a prototype and a real developed application would need a completely rewritten server part anyway as it was not meant to support more than one client.

8.3.1 Programming Language Choice

While experimenting with the BART algorithm I used the Python programming language as I felt it was easier and faster for prototyping. Only the client part (the *sender* in the BART model) was later translated to Java as it has to run with the Java client and without the Python language environment. I

opted against using *Jython* (Python code executing in the Java virtual machine) as the mathematical library *numpy* was not ported to *Jython* and it would violate the agreement that the prototype diagnostic program was to be made in Java (and not only run in its virtual machine). But the server part (the *receiver* in the BART model) was not translated because of too little time remaining of the project and little gain in return.

8.3.2 Implementation of the BART Algorithm

The algorithm could be implemented simply as described in the paper, with one way communication. But due to problems of network address translation (NAT) and blocked TCP and UDP ports for normal internet users I chose to use the centrally located server as the receiver in the BART method. Then, via another channel, it can report back to the client with the bandwidth result. To this end the client first establishes a TCP stream socket to communicate messages for greeting, traffic rate, ready-to-receive and bandwidth result.
The UDP probe packets in the actual BART algorithm are constructed with a header of 28 octets, described in *Table 8-3*. The content of the rest of the packet is not specified.

Table 8-3. BART probe packet header definition.

4-octet block	Contains	Comment
1	0x62617274	Is ASCII character codes for 'bart', used as header identifier
2	0x62617274	As above
3	0x62617274	As above
4	0x62617274	As above
5	sequence ID	Randomized ID of this train
6	traffic rate	The inter packet spacing between sends, in milliseconds (not really used in this prototype, instead communicated via the stream socket channel)
7	packet #	Identifies this packet in this particular train

8.3.3 Problems with the BART algorithm

There is no example code published. The way to implement it was to read and re-read the scientific paper to understand it. But there is always some little fine tuning that is left out and one needs to reinvent, which makes it hard to perfect it for production use.
The other kind of problem is that it is now an intellectual property of Ericsson where some of the inventors work. To use it commercially one should acquire a commercial license from Ericsson [8].

8.4 Java Native Interface

To have a central facility for handling the native interface operations a special package was created. The sub package for native networking exports an interface to plug-ins to use. There is not any support for using it concurrently, i.e. multiple plug-ins should not be run at the same time; since this can create undefined behaviors as not all functions within the package are stateless. Within this package the operating system differences are hidden from the plug-ins.

8.4.1 Choice of language

JNI has direct support for C, C++ and assembly language. As the latter is overly complicated I ruled it out directly and then I chose C over C++. For me C++ has too many features so the language is too big and not coherent in its design. This code was not going to be big, and definitely did not need any object orientation, so C was the best choice.

8.5 Encountered Impediments

This section describes different obstacles that were encountered during the development process, things that had to be overcome without changing the general development path, or in some case resulted in abandoning a feature already in development.

8.5.1 Java

While in search for facilities to do network related programming in the Java standard library it turned out to be a disappointment as the prototype does not only need normal network communication, but also network configuration data in the operating system.
Java has a limited capability to retrieve the IP address of the machine, it only returns 1 address, if available, although multiple adapters with multiple addresses can exist.
Also there is no way to find the operating system's routing table, useful for determining the default gateway.

Java does not have support for communicating via the ICMP protocol directly. In the `Java.net.InetAddress` class there is a new function, introduced in J2SE 5.0, called `isReachable`. But this does not live up to the need because of limitations in the platforms' implementation. Sun's JRE on Solaris tries with ICMP first if the required privileges exist (need to act as the *root* user, that is not normal) or falls back on TCP port 7 (the echo port) otherwise. But the later is often stealthily blocked by firewalls, meaning that the host doesn't give any response at all back when another host tries to establish a connection. Sun's JRE for Windows does not even have ICMP echo request built-in, it just tries the TCP variant. Apart from this major problem there is also the inability to measure the round trip time. Some Windows versions take some time

before they respond with a "connection refused" message when there is no program listening on a port even when the port is not stealthily blocked.

With all these limitations, it turned out that the only hope to do this network connection diagnostic was to do it with platform dependent native code using the *Java Native Interface* (JNI).

Using the native interface there is a need to have platform specific dynamic libraries that the JRE loads into its memory space. The JAR file needs to contain them all and extract the appropriate version depending on the running operating system and JRE data model. Here there is a problem; there is no direct way to unload a library from memory and while it is being used the dynamic library file cannot be deleted.

Another, not so obvious, problem is that the Sun's JRE now is available in a 64 bit version. The operating systems do not support loading 32 bit dynamic libraries into a 64 bit process's memory space.

8.5.2 Windows

A problem with Microsoft is that they are not aspiring to get their compiler (the one used in Visual C++) fully C99 compliant, citing lack of interest by the users [9]. They have done some parts of C99 requested by users but the *stdint.h* header file is still not shipped with the compiler even though it would not require any changes to the compiler itself. There are 3rd party versions of *stdint.h* available for Visual C++, I used one such [10]. The catch is that a copyright notice, conditions and a disclaimer of liability should be reproduced if the program that uses it is distributed. I have not included such a notice as I don't expect this prototype be distributed in its current version.

8.5.3 Mac OS X

The main difficulty for me was the documentation for developers. Apple's developer site was hard to grasp, to know where things could be found. It had both the manual pages from the UNIX heritage and Apple's different frameworks. It would have been easier to find things if the content was grouped together in a way that allows the searcher to look for a topic, not the place where it can be found. E.g. network topics, the manual pages had their topics and Apple had their own network framework documentation.

8.6 Rejected development paths

During the planning of the prototype I read material and also made small test programs in advance. These things helped in decisions to abandon some initial ideas for the prototype. Here the major ones are presented with rationale for rejection.

8.6.1 Deviation from the Thesis Specification

While working on this thesis I came to realize that the network diagnostic part of the specification is the most interesting and other parts more routine work. Instead of doing step 5 (non-network tests) and 6 (collect information to customer support service) from the thesis specification I (with necessary approval) did the work on measuring the bandwidth between two hosts, using the BART algorithm, which added more interesting scientific value to this thesis and more integrated with the theme of the first big part – testing the network connection.

8.6.2 Operation System Support

As described previously Java does not have any good support for lower level network programming and instead native code has been written. On the Linux platform there seems to be no easy solution for sending ICMP echo requests without being root and having access to raw sockets. Even the normal ping command requires root privileges but has a *setuid flag* to elevate normal users when running that program. As this is just a prototype and the number of inexperienced computer users (targets) running Linux is very small the support for that operating system was dropped and instead we focus more on network probing. If needed in the future a Linux native backend with raw sockets running the JRE as root would make this prototype work equally as well on Linux.

9. Testing

As with any type of software it is not enough to code it, one must also know that it works as intended. But, doing quality assurance testing on a prototype for a master thesis is going too far (unless of course the thesis is about software testing). The application has only been tested to run on the latest version of SUN's Java runtime environment in J2SE 5.0 branch, on Windows XP, Windows 7 and OS X 10.5.

9.1 Source Code Testing

Some parts of the code have test cases that can be run to verify that those modules work as intended. This utilizes *JUnit* unit testing framework and the *EasyMock* mock object framework.

10. Future development

This section lists smaller things that were not completed for the prototype but should be if the code will be developed towards a final product.

- For the program to run correctly in 64-bit JRE it also has to include 64-bit versions of dynamic linked library files for each such platform. Currently only 32-bit is used.

- The OS X has support for detecting if the network medium is present or not, but I did not found out how to use it until later when I had already skipped that feature for OS X. The information can be retrieved via IOCTL system call. That is how *ifconfig* shell command does it. Studying the nearly uncommented open source code for *ifconfig* and understanding it probably gives the solution.
- According to a web page [11] the solution to the problem with being unable to delete the native dynamic loaded library file is to write a custom Java class loader, use it to load the native library. Later at shutdown all references to objects of the native class should be dropped and also to this class loader. Running the garbage collector twice after that was claimed to remove the file lock and make deletion possible.
- Use native libraries to get all IP addresses for all network interfaces. Then keep track of them and when the default router address is found that IP address should be checked to be within the same network address part of at least one of the previously found host IP addresses.
- This program is not a shell script and thus should not rely on executing system commands and parsing the output. One such case where that rule is broken is for finding the name servers (DNS) on OS X. The *scutil* command is used, but the System Configuration framework API, SCDynamicStore, can be used to do queries to the system about various runtime parameters instead.
- Optimize some network tests to be done in parallel. E.g. the game server test is not sensitive in measuring time, it just needs to open a number of stream socket sessions which are totally independent of each other. If there will be a great number of servers to check this can cut down time for tests to complete, in the case where some servers timeouts.
- The user interface of the start window was intended to contain a list of available test & tools. But the list is quite short (two enabled and two disabled mockup buttons) so it could be expanded to have all shown at the same time instead of a scrollable surface.
- For a live release it needs a possibility to be branded to different operators on the poker network, perhaps by detecting which operator's client is installed and show that brand.
- Fine-tune the bandwidth testing so it produces more stable and reliable results.
- Improve the router testing in the network test by doing a trace route to some location and thus finding out the second router (hop). Often people have a home router. The next hop is often at the ISP and thus it can be used to detect if a user has a local problem with the ISP.
- Verify the content of the DNS response to make the test not only detecting that the DNS responds but it also has contact with the rest of the DNS infrastructure (i.e. it can connect to other DNS servers).

11. Results

11.1 Built-in diagnostic in operating systems

Compared to 8 years ago the situation has improved from nearly nothing to quite good regarding automatic network diagnostics. But they are very specific to what the manufacturer thinks should be tested (except Windows 7 where one can write one). Here is where tools made by the poker provider would fit in as they want to customize connection tests to their software etc.

11.2 Prototype program

11.2.1 Network Connection Test

Albeit it could produce inconsequent results as mentioned in the previous section about mismatches of host IP and gateway IP addresses I think the network connection test works well. But the graphical interface is not so well designed, lacking human computer interaction user perspective thinking).

11.2.2 Bandwidth Test

This test was not completed as I could not get my implementation to produce any reliable output. That is because of the problem with getting it fine-tuned to work well and lack of a big real world test to verify that it works. It is possible to complete it but for any commercial release it should be considered that it is intellectual property of another company, which perhaps can license it to bwin.

11.2.3 Graphical User Interface

As a mockup the Ongame network logo is used in the main window instead of any operator's brand, no need for that in a prototype.

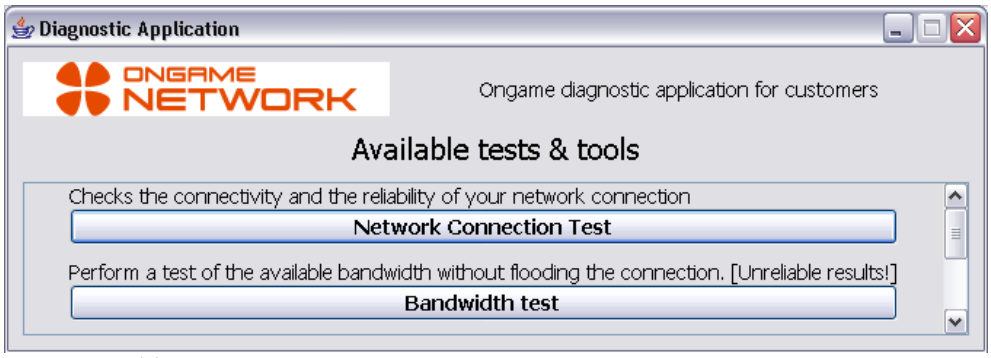

Figure 11-1. The main window.

Clicking the button labeled "Network Connection Test" in *Figure 11-1* takes the user to the screen shown in *Figure 11-2* where the actual test starts right away. The first three tests can be done in an instant as they are simple lookup requests to the local computers configuration. The fourth test is also usually done without seeing it, as the DNS server is often very close (in context of

time) to the host. The fifth test of reachability for internet ping sites takes a longer time as the program waits a little while between each ICMP echo response and the next ICMP echo request.
If any of the basic tests (the checkboxes in *Figure 11-2*) fails, the text area in the bottom of the window will show a (hopefully helpful) description of the probable problem.
When the test finishes a "button" (a symbolic icon, not interactive GUI element) is shown, together with a short text about the connection quality. This is shown in *Figure 11-3*. The button color reflects the result of the quality test, as people are used to interpreting signs using colors. Red (no connection) is often used to signal a warning and green (perfect connection) is common to signal that something is okay. The button's colors spans from red to green via yellow in six steps.

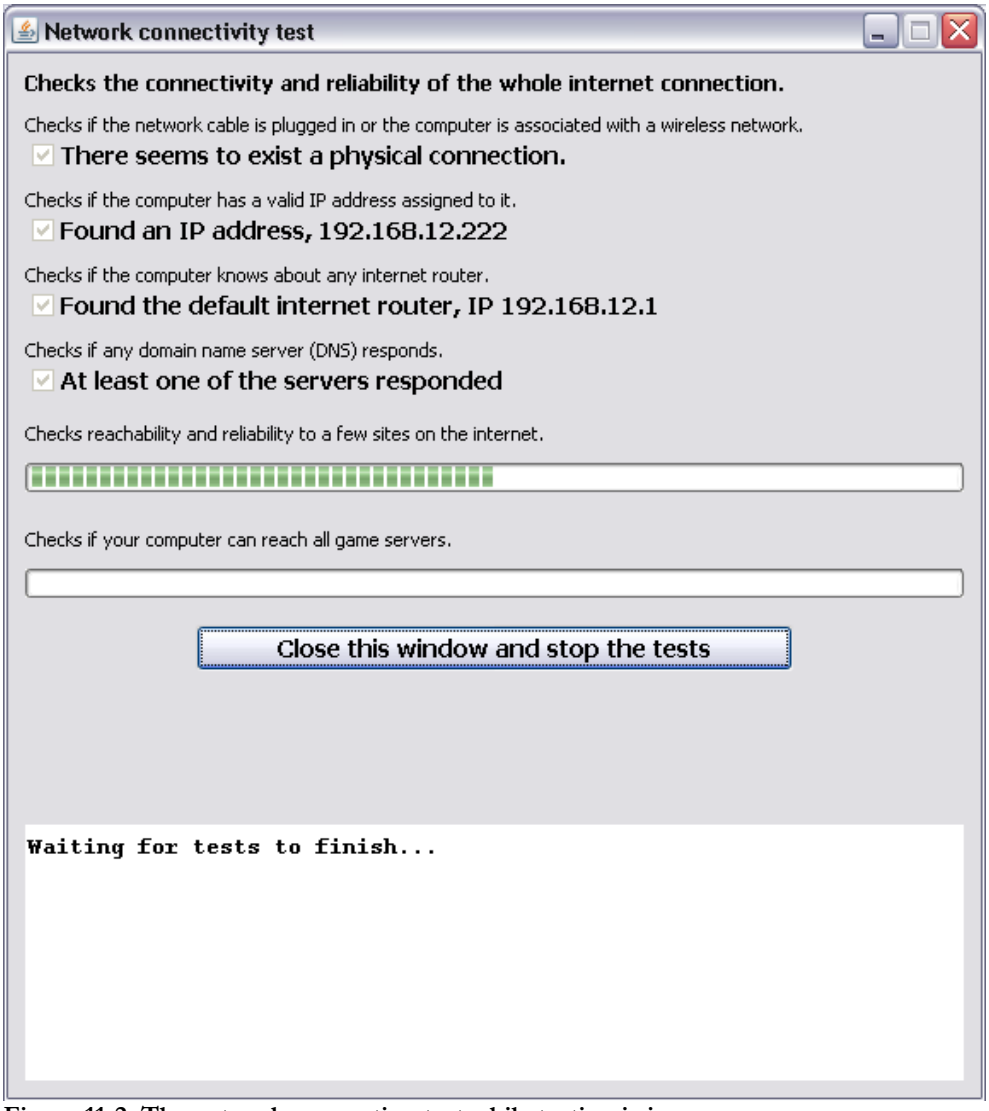

Figure 11-2. The network connection test while testing is in progress.

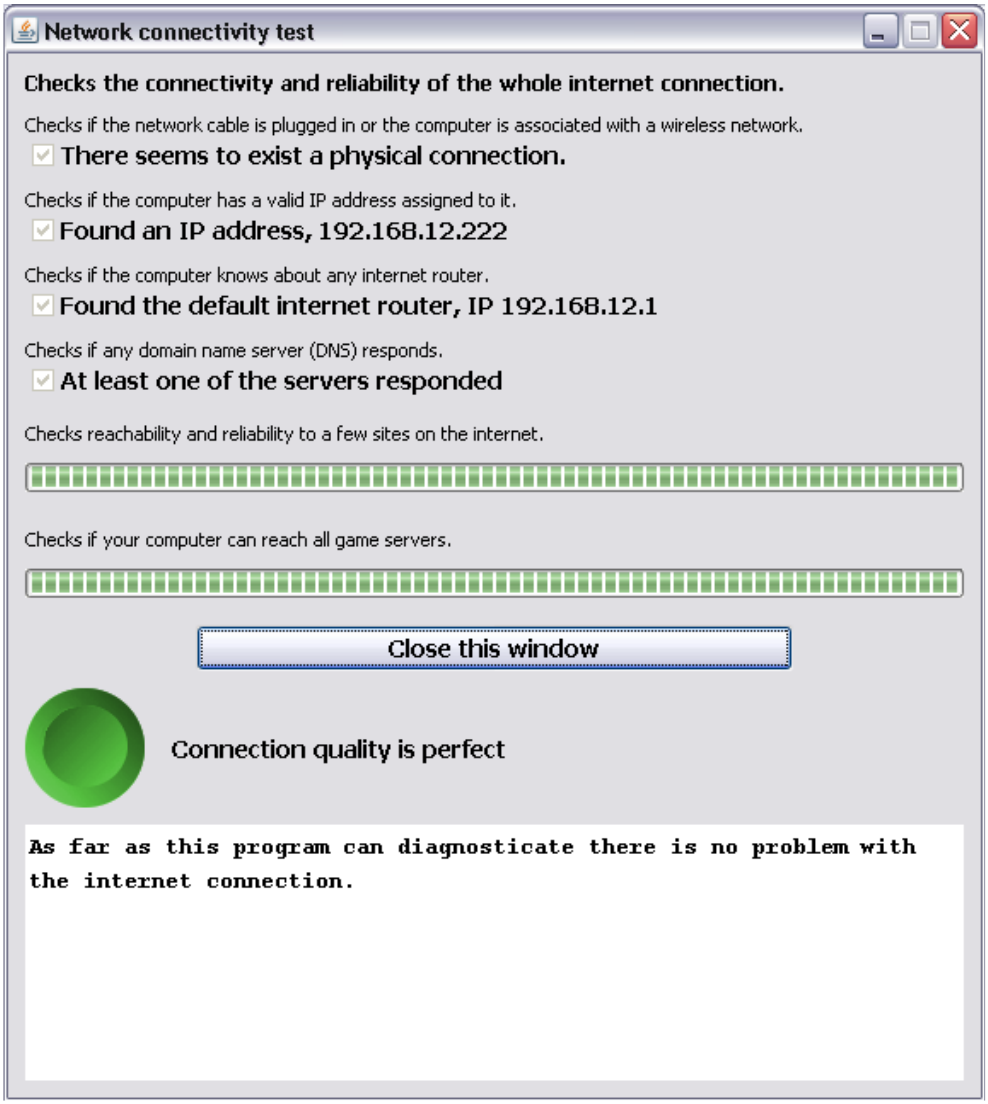

Figure 11-3. A connection test completed with perfect score and a green button.

11.3 Implementation

Before the thesis started I thought that it could be done with Java. It turned out that nearly everything about network testing needs to be done in native code for each platform.

In this section the source code is described.

11.3.1 Java Source Code

The package names are listed in *Table 11-1*.

Table 11-1. List of Java packages and their content.

Package name and search path	Description
`com.bwin.network_diagnostic_tool`	Root package. Contains startup code, some interfaces and handler code for test & tools

`.bandwidthtest`
&
`.bandwidthtest.gui`
	The experimental code for BART testing and its GUI.

`.gui`	The GUI for the main window.

`.jni`
&
`.jni.net`
	The Java Native Interface code. As all JNI code relates to networking the base .jni packages is empty of functional code.

`.networktest`
&
`.networktest.gui`
	The network connection test with its GUI.

`.test.networktest`	Package with code to run tests, does not contain any production code.

11.3.2 Common Platform Dependent Source Code

The DNS test works by doing a real query to the DNS server, not using ICMP echo request. The reason to implement it "by hand" was to be sure that it is really the DNS server that responds and not a DNS lookup cache if a standard Java library function like `java.net.InetAddress.getByName` is used. Also it required manipulation of specific bits in the construction of the packet header, which is a lot easier in C than in Java (as the later lacks unsigned types).
As both Windows and OS X uses BSD style sockets the source code for actually transmitting and receiving was very similar. So similar that it was easier to build it in one file with a few preprocessor directives like `#ifdef ... #endif` to select which code to compile where it differed between the platforms.

Table 11-2. The exported C function that is common in the native classes NativeWin32 and NativeOSX from the java package com.bwin.network_diagnostic_tool.jni.net.

Function	Description
`testDNS`	Constructs a UDP datagram that is to be sent to a DNS server requesting it to look up an address. Does not care about the content of the response, only that a response actually is received from the server.

11.3.3 Windows Native Code

The windows native code was easy to write thanks to a good support from the Windows API and good and comprehensive documentation at MSDN (Microsoft Developer Network) webpage. Once the appropriate API and function had been found it was more or less a matter of simple adapting it. It required loading other Windows .dll files dynamically, so it has a "startup" and "shutdown" function that has to be run first and last. Table 11-3 shows the C functions that are programmed with JNI to be available to the native class for Win32 available to the Java source code.

Table 11-3. List of exported C functions that are exposed in the native class shown in java source tree as com.bwin.network_diagnostic_tool.jni.net.NativeWin32.

Function	Description
`Startup`	Performs a operating system version check and loads dynamic libraries for ICMP, IPHelper and SENS.
`Shutdown`	Closes handles and frees dynamic library loadings.
`icmpInitiateEchoRequest`	Create an ICMP echo request handle. Needs to be run once
`icmpLockTargetEchoRequest`	Specifies IP address of the host to be 'pinged'.
`icmpSendEchoRequest`	Transmits the ICMP echo request to the previously specified IP address.
`sensIsNetworkAlive`	Is responsible for checking if there is any network connection alive at all.
`iphelperGetBestRoute`	Extracts the default gateway IP address.
`iphelperGetNetworkParams`	Extracts a list of DNS servers.

11.3.4 OS X Native Code

The OS X code was harder to write because it does not provide as many things in the API. E.g. to send ICMP echo requests in Windows it was simply a matter of providing address, send buffer, receive buffer and timeout value. In OS X it required to open a special datagram socket for ICMP (not UDP) and constructing the ICMP package "manually" and processing the reply (a full IP package) "manually". *Table 11-4* shows the C functions that are programmed with JNI to be available to the native class for OSX available to the Java source code.

Table 11-4. List of exported C functions that are exposed in the native class shown in java source tree as com.bwin.network_diagnostic_tool.jni.net.NativeOSX.

Function	Description
`icmpPrepareEchoRequest`	Open a datagram socket targeted at the host to be 'pinged' and allocates buffers.
`icmpSendEchoRequest`	Transmits the ICMP echo request to the previously specified IP address and takes care of interpreting the answer (A whole IP packet).
`icmpTerminate`	Frees the buffers related to ICMP echo requests.
`getGateway`	Retrieves the default gateway address.
`getDNSservers`	Retrieves a list of DNS servers.

11.4 User feedback on the prototype

I think developing something without asking people to really give feedback is not such a good idea. Here are some thoughts presented from two parties, the customer service that of course have a interest in this and a friend of mine who is 2nd year master student in human computer interaction. In HCI user friendliness is important and especially the graphical interfaces.

11.4.1 From Agents at 2nd line customer service department

Generally the agents agreed that this kind of tool would be best to bundle with the downloadable poker client but also have a web based variant (perhaps with a signed Java applet).
There were no remarks on the simple user interface.
Question about the bandwidth test was raised, what benefit it has over websites like speedtest.net (bandwidth tests were discussed in section 5.3).

11.4.2 From Sohail Sahab

Here is the valuable input from Sohail divided between the main window and the network connection test window.
Main window:

- The scroll bar in the main window is not correct to use at all as there are so few items (tests & tools) on display.
- There is no distinct graphical division between the different tests listed.
- The test's or tool's name should not be put into a big wide button.
- The logo takes up quite a lot of space.

Network connection test:

- There should always be some indication of progress when the window is opened. In the case where it failed on the first test the user would wonder if something will happen even though text is displayed that there is no working connection.
- The checkbox GUI element should not be used, instead green or red marks for every subtest that now has a checkbox as indicator.
- The symbol button for the connection quality is not good as users instinctively want to press on buttons. As a solution that the quality symbol does not only have two state (red or green) Sohail suggested a gauge from red to green via yellow and a pointer showing the current quality of the connection.
- Again there could be wider margins to the borders of the window, the text and progress bars do not need to be so wide. The text for each subtest could have a larger indentation and the spacing between them should be bigger.
- All fonts should have bigger sizes so the smallest one is not as small as now (it has the standard Windows' font size).
- The quality symbol button (and any replacement gauge) should be centered in the window, not be on the side.

Sohail also got to the see the main window of the GUI mockup programmed in C# that I did in the beginning of the development, see *Figure 11-4*. He thought it was much better as there were icons for each test (albeit not everyone had a good correspondence with its purpose) and the button to start it was clearly a button to start the test and not the whole name of the test within. Also the graphical separation between the tests was also good.

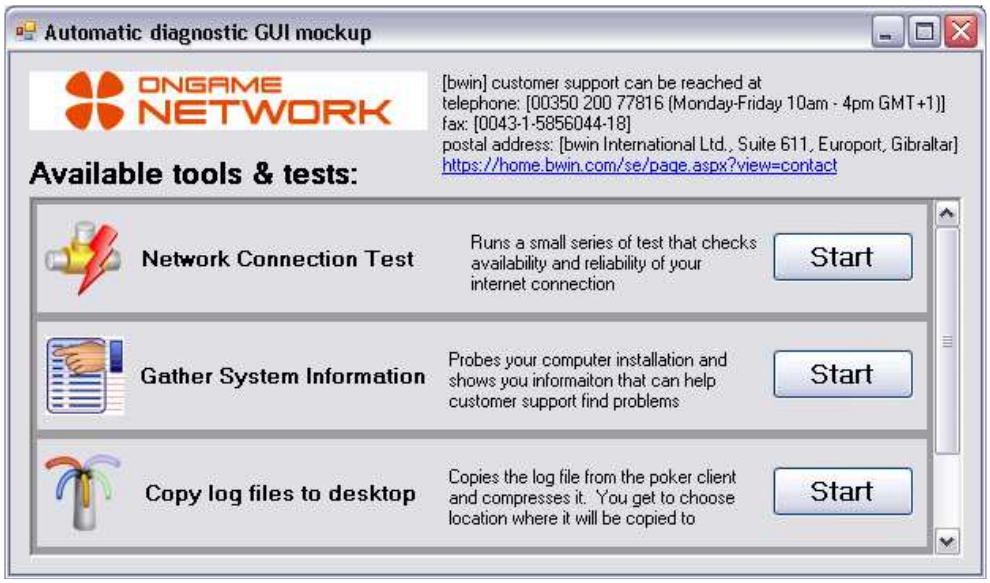

Figure 11-4. Early mockup envisioning the GUI.

12. Discussions

12.1 To be, or not to be

As this prototype application was possible to construct, albeit needing more work, in this time by one student it can be compared to the working time and money spent by customer service for assisting with these repetitive request. Also it must be considered the loss of income while customers are not able to play. Commercially it would in my opinion be a good bargain to develop this product and take it to another level so it is shipped with the poker client software. But it should be expanded to feature more tests and tools that are focused on the poker software in use. As demonstrated modern operating system (Windows 7 and OS X) are distributed with fairly good diagnostic and repair tools, thus making a program that only works with network problems a bit redundant. But it will take a long time before Windows XP will have negligible market share compared to Windows Vista and Windows 7.
It seems that a program, such as this, is useful for detecting network problems and presenting probable causes to the user, but it would be best to leave the automatic network repair to the operating systems, built-in features which can be launched by this diagnostic program if any network error is detected.

If the computer does not get onto the internet it will be some extra work for the customer to bring the program to the computer. Hence a Java web start application should not be the only distribution model. But integrating these tests and tools into the same executable file as the poker client is probably a bad idea. It can happen that the problem is that the poker client software fails to start, thus preventing the launch of the diagnostic tool. Bundling the software as a stand-alone executable and also making it available via Java web start could be the best solution.

12.2 Self-evaluation of the prototype

In retrospect I can think of a few things I would have liked to give more focus during the development. The interface could have been done better and I could have asked for input earlier in the development so it would have been time to change it. Secondly it would have been interesting to have found out about BART algorithm earlier and then I could have more time to develop it.
I don not regret that I have left out small parts instead of perfecting every part, because it is after all a prototype to give a suggestion of what could be created.

12.3 Future features

When it is not a network problem it can often be a computer configuration problem. A good tool in the program for retrieving all sorts of information, and with the customers consent, sends it to the customer service would be of good use to find patterns to problems that are not frequent and hard to define the cause of. In this specific version of the poker client the log files are hard to

find, as described in the section "6.2 Location of log files in the file system". A tool that simple copies them to the computer's desktop and compresses them for the user to e-mail to customer service would greatly speed up problem solving time.

13. References

All links have been verified to work on January 11, 2010, before printing occured.

[1] Sherwood Lawrence, "*Network Diagnostics Tools Feature Overview*", 2001, http://technet.microsoft.com/en-us/library/bb456996.aspx#EEAA (direct link to section *Network Connections Repair Link*)

[2] Microsoft knowledge base article 914440, "*Network Diagnostics for Windows XP is available to help identify and fix network connection problems*", November 19, 2007 - Revision: 2.3, http://support.microsoft.com/kb/914440

[3] Svante Ekelin, Martin Nilsson, Erik Hartikainen, Andreas Johnsson, Jan-Erik Mångs, Bob Melander and Mats Björkman, "*Real-Time Measurement of End-to-End Available Bandwidth using Kalman Filtering*", 2006, http://baker.ilik.net/~svante/evergrow/RTME2EABKF_NOMS_2006.pdf

[4] pathChirp homepage, http://www.spin.rice.edu/Software/pathChirp/

[5] Greg Welch and Gary Bishop, "*An introduction to the Kalman filter*", 2001, http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html

[6] Microsoft knowledge base article 310294, see sections "*Classify Application Data"* and *"Store Application Data in the Correct Location*", March 20, 2008 - Revision: 3.1 http://support.microsoft.com/kb/310294

[7] IETF RFC 1035, "*Domain names - implementation and specification*", 1987, http://www.ietf.org/rfc/rfc1035.txt

[8] Page about licensing on the homepage about BART, http://www.barttool.org/licensing/

[9] Arjun Bijanki at the Visual C++ Team Blog, blog post "*ISO C Standard Update*", 2007, http://blogs.msdn.com/vcblog/archive/2007/11/05/iso-c-standard-update.aspx

[10] Alexander Chemeris' *stdint.h* header file, http://msinttypes.googlecode.com/svn/trunk/stdint.h

[11] A discussion thread at SUN's forums about JNI. Read first answer (posted 2008-aug-07 11:57) to the question, title of thread is "*Java Native Interface (JNI) - loading the shared library without using system.loadLibrary();*", 2008, http://forums.sun.com/thread.jspa?threadID=5321076

APPENDIX A:
Automatic error diagnostic for software and network connection problems

1. Background

Determining the cause of a malfunction of a software service can be difficult and time consuming if the support personnel can't access the affected computer directly, especially if giving support over telephone with a novice computer user.
This kind of support is normal when the product/service is a one-time payment and the company has gotten the money already. For an online poker company like bwin it's worse, if the customer can't play they can't spend money and generate revenue for the company and thus it's critical that problems are swiftly fixed.

2. Task description

STEP 1

Investigate how network problems are diagnosed and possibilities to diagnose different operating systems (e.g. Windows, Mac OS and Linux) settings.

STEP 2

Survey what wishes the support and other departments at bwin have on this kind of tool and if necessary take those into account if possible in the next step.

STEP 3

Investigate what kind of network problems that are the most common when the game client can't reach the game server. Are there any specific problems with bwin's own protocol? Can these problems be circumvented?

STEP 4

Implement a proof-of-concept signed Java Web Start (JNLP) program that seemingly intelligently finds the cause of a problem with the (or lack thereof) connection to bwin's various servers (game, web, patch, etc.) and the local software installation. The program should be created to allow easy extension, probably a plug-in architecture for tests and presentations. The program

should inform the customer what is to be diagnosed. Use information gained from the first step to prioritize which of these tests should be made.
Proposed general tests:

- Can the program access the network (thus checking a software firewall)?
- Can the local gateway be reached?
- Run a trace of the routing (like traceroute/tracert program) to see where the packets are dropped outside of the local network.

The result should be presented in a form comprehensible for a customer with average computer knowledge and that they over telephone or mail can explain to the support personnel.

STEP 5

When the prototype works (can handle the tests in step 4), extend it with more tests that are not only network related. Proposed tests:

- Gather information about the software environment, e.g. operating system, web browser and java runtime environment.
- Find information about the bwin poker client and use bwin's own client-server protocol to test communication more than just reachability of servers.

STEP 6

A further step if time is available can be to do a server backend that the diagnostic program can send its reports to with the customer's permission. Also a communication from server to client could be made so general reports from the support personnel can be publicized in the diagnostic program. This would be useful to inform users of interruptions in the service that the support is already well aware of (like communication from a specific country is problematic at the time).

3. Procedure

RELATED COURSES

The thesis does not target a single field of research but the course material closest to this would be network communication. Other courses that have contributed with knowledge are e.g. methods of programming DV2 in writing good code and using tools to correct and enhance the code, software engineering to understand software development project and the CS project course in practical working of a (relatively) big project.

LITERATURE

Any big good book on network communication will be useful.

DEVELOPMENT TOOLS

IntelliJ IDEA for developing with Java.

DEVELOPMENT METHODOLOGY

Not working in team but scheduling oneself with tasks based on how it's done with scrum post-its.

EVALUATION OF PROJECT

The final report should give a good view on how network problems have been investigated, technical description of which causes they have, the effects for the customer and the solution (if any).
It should have design documentation for the prototype developed during the project and remarks on how the prototype (most probably) differed from the initial design thoughts.

4. Delimitations

The prototype should not be made to by itself repair problems, like reconfiguring settings in the customer's computer etc. While it would be useful the thesis is already big enough.

5. Time schedule

- Preparations: 1 week.
- Step 1 – investigate how to diagnose: 1 week.
- Step 2 – survey wishes: 1 week.
- Step 3 – investigate current problems: 2 weeks.
- Design of system: 2 weeks.
- Implementation; 9 weeks split among these bullets
    - Basic skeleton with extensibility support.
    - Network diagnostic tests.
    - Local software information tests.
    - Server part if time allows.
- Writing report: 4 weeks.